Strategy

Intent

- define a family of related algorithms, encapsulate each one, and make them interchangeable by providing a common interface
- algorithm may vary by object and also may vary over time

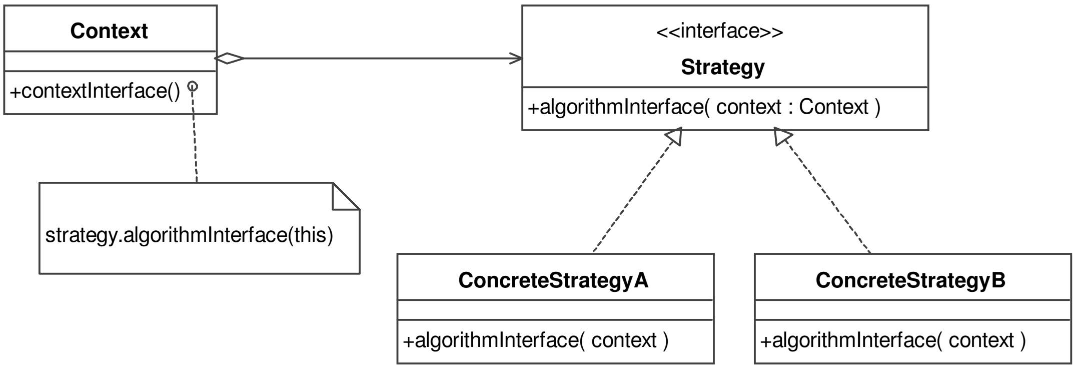
Motivation:

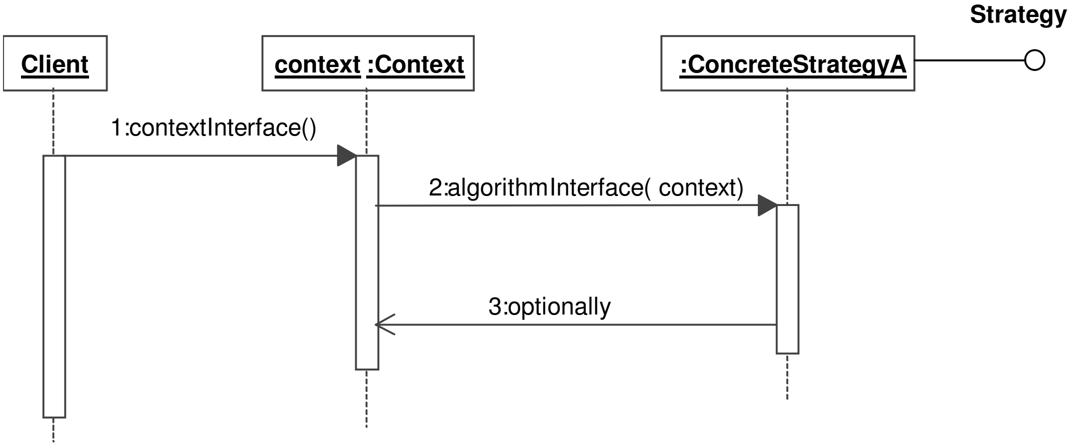
- many different algorithms exist for fulfilling a responsibility
- You need different variants of an algorithm
- An algorithm uses data that clients shouldn't know about
- A class defines many behaviors, and these appear as multiple switch statement in the classes operations
- Many related classes differ only in their behavior
- you want to separate the algorithms from the object which is using them because
  - an object may decide to change the algorithm it uses
  - an object will get complex if you include all the code of the algorithms
  - an object will have to use conditional code to decide which algorithm it will use
  - each algorithm uses its own algorithm-specific data
  - it is difficult to add new algorithms if they are hard coded in the object.

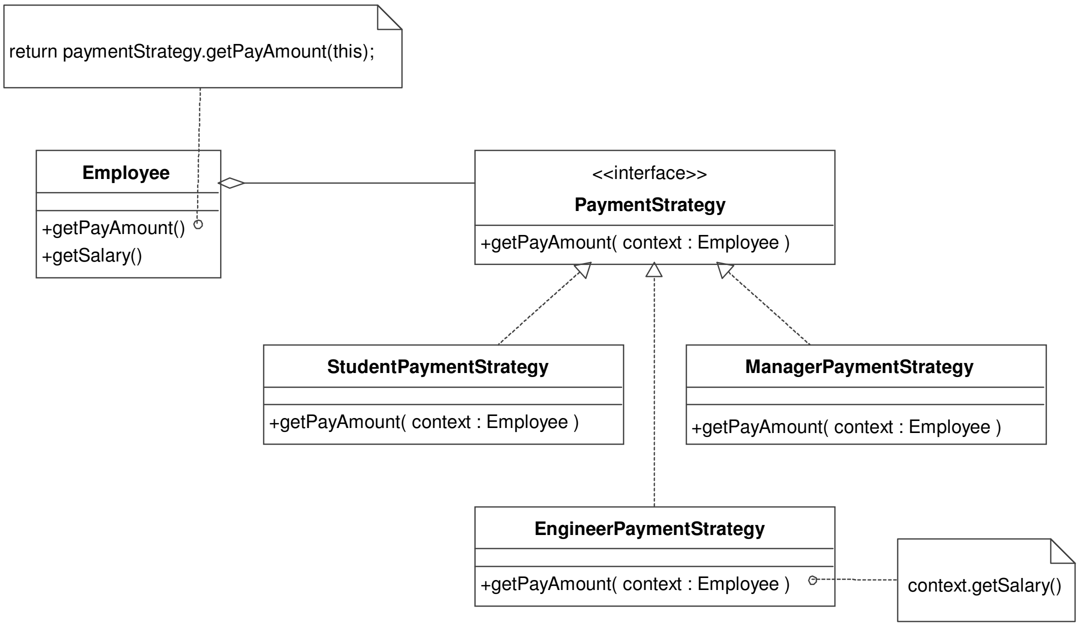
Problem

- Conditional statements appear in context object determining which algorithm to use

Solution


Context
+contextInterface()
strategy.algorithmInterface(this)
<<interface>>
Strategy
+algorithmInterface(context : Context)
ConcreteStrategyA
+algorithmInterface(context)
ConcreteStrategyB
+algorithmInterface(context)




return paymentStrategy.getPayAmount(this);
Employee
+getPayAmount()
+getSalary()
<<interface>>
PaymentStrategy
+getPayAmount(context : Employee)
StudentPaymentStrategy
+getPayAmount(context : Employee)
ManagerPaymentStrategy
+getPayAmount(context : Employee)
EngineerPaymentStrategy
+getPayAmount(context : Employee)
context.getSalary()

Exercises

- Modify the code, so that we can change the payment strategy of an employee.
- Add a **getBonus** method to employees. Employees might get a bonus based on their salary (a percentage of the salary different for each employee), or a fixed bonus (different for each employee).